NASA/TM-2011-217152

Model Checking a Self-Stabilizing Distributed Clock Synchronization Protocol for Arbitrary Digraphs

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

May 2011

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, and organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at 443-757-5803

- Phone the NASA STI Help Desk at 443-757-5802

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/TM-2011-217152

Model Checking a Self-Stabilizing Distributed Clock Synchronization Protocol for Arbitrary Digraphs

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

May 2011

Acknowledgments

This effort was conducted under the Integrated Vehicle Health Management (IVHM) project of NASA's Aviation Safety program. The author would like to thank the reviewers for their in-depth reviews and constructive comments.

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Abstract

This report presents the mechanical verification of a self-stabilizing distributed clock synchronization protocol for arbitrary digraphs in the absence of faults. This protocol does not rely on assumptions about the initial state of the system, other than the presence of at least one node, and no central clock or a centrally generated signal, pulse, or message is used. The system under study is an arbitrary, non-partitioned digraph ranging from fully connected to 1-connected networks of nodes while allowing for differences in the network elements. Nodes are anonymous, i.e., they do not have unique identities. There is no theoretical limit on the maximum number of participating nodes. The only constraint on the behavior of the node is that the interactions with other nodes are restricted to defined links and interfaces. This protocol deterministically converges within a time bound that is a linear function of the self-stabilization period. A bounded model of the protocol is verified using the Symbolic Model Verifier (SMV) for a subset of digraphs. Modeling challenges of the protocol and the system are addressed. The model checking effort is focused on verifying correctness of the bounded model of the protocol as well as confirmation of claims of determinism and linear convergence with respect to the self-stabilization period.

Table of Contents

1. Introduction

Synchronization algorithms are essential for managing the use of resources and controlling communication in a distributed system. **Synchronization** of a distributed system is the process of **achieving** and **maintaining** a bounded skew among independent local clocks. A distributed system is said to be self-stabilizing if, from an arbitrary state, it is guaranteed to reach a legitimate state in a finite amount of time and remain in a legitimate state. A legitimate state is a state where all parts in the system are in synchrony. The self-stabilizing distributed-system clock synchronization problem is, therefore, to develop an algorithm (i.e., a protocol) to *achieve* and *maintain* synchrony of local clocks in a distributed system after experiencing system-wide disruptions in the presence of network element imperfections. The **convergence** and **closure** properties address achieving and maintaining network synchrony, respectively. Hereafter in this report, we use the term synchronization to mean self-stabilizing clock synchronization in distributed systems.

A thorough understanding of the synchronization of a distributed system has proven to be elusive for decades. The main challenges associated with distributed synchronization are the complexity of developing a solution and proving the correctness of these solutions. It is possible to have a solution that is hard to prove or refute. Such a solution, however, is not likely to be accepted or used in practical systems. The proposed solutions must restore synchrony and coordinated operations after experiencing system-wide disruptions in the presence of network element imperfections and, for ultra-reliable distributed system, in the presence of various faults. A fault is a defect or flaw in a system component resulting in an incorrect state [Gir 2005] [Tor 2005] [But 2008]. In addition, a proposed solution must be proven to be correct. If a mathematical proof is deemed difficult, at a minimum, the proposed solution must be shown to be correct using available formal methods. Furthermore, addressing network element imperfections is necessary to make a solution applicable to realizable systems.

Typically, verification of a protocol is conducted by the composition of a paper-and-pencil proof. Verification of such proofs is another challenge associated with self-stabilization, especially as the complexity of the protocol increases. Such proofs are error prone.

In [Mal 2011] a solution is presented for an arbitrary network (digraph) in the absence of faults. The system under study is an arbitrary, non-partitioned digraph ranging from fully connected to 1-connected networks of nodes while allowing for differences in the network elements. Some networks of interest include grid, ring, fully connected, bipartite, and star (hub) formation. This solution does not require any particular information flow nor imposes changes (e.g., embedding a directed spanning tree or rewiring) to the network in order to achieve synchrony. The assumption of an absence of faults is equivalent to the assumption that all faults are detectable. This departure from our previous work at the Byzantine extreme of the fault spectrum [Mal 2006] is in part because of the niche use and the extra cost associated with the Byzantine faults. Also, using authentication and error detection techniques, it is possible to substantially reduce the effects of variety of faults in the system. Furthermore, the classical definition of a self-stabilizing algorithm assumes generally that there are no faults in the system.

In this report we present model checking efforts in support of the claims of *A Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs* [Mal 2011]. In particular, this effort encompasses the verification of correctness of a bounded model of the protocol by confirming that a set of candidate systems self-stabilizes from any state. This effort, furthermore, includes the verification of claims of determinism and linear convergence of the bounded model of the protocol with respect to the self-stabilization period. Toward this objective, a number of abstractions and reduction techniques are devised to reduce the state space. The model checking results of the bounded model of the protocol have validated the correctness of the protocol as they apply to the networks with unidirectional and bidirectional links. In addition, the results have confirmed the claims of determinism and linear convergence.

The following sections describe the model checking efforts in detail. In section 2 we provide a system overview. We present the protocol and its description in section 3. Modeling specifications and abstractions used in describing a bounded model of this protocol are described in section 4, where the underlying topology and network models are defined. In section 5 we enumerate the propositions used and, finally, in section 6, we present a summary of the model checking results and concluding remarks.

2. System Overview

We consider a system of pulse-coupled entities (e.g., oscillators, pacemaker cells) pulsating periodically at regular time intervals. We model the system as a set of nodes that represent the pulse-coupled entities and a set of communication channels that represent their interconnectivity. The underlying topology considered here is a network of $K \geq 1$ nodes that exchange messages through a set of communication channels. Nodes are anonymous, i.e., they do not have unique identities. All nodes are assumed to be good, i.e., actively participate in the synchronization process and correctly execute the protocol. The communication channels are assumed to connect a set of source nodes to a set of destination nodes with a source node being different than a destination node. All communication channels are assumed to be good, i.e., reliably transfer data from their source nodes to their destination nodes. The nodes communicate with each other by exchanging broadcast messages. Broadcast of a message by a node is realized by transmitting the message, at the same time, to all nodes that are directly connected to it. The communication network does not guarantee any relative order of arrival of a broadcast message at the receiving nodes, that is, a consistent delivery order of a set of messages does not necessarily reflect the temporal or causal order of the message transmissions [Kop 1997]. There is neither a central system clock nor an externally generated global pulse or message at the network level. The communication channels and nodes can behave arbitrarily provided that eventually the system adheres to the protocol assumptions (Section 3.4).

2.1. Drift Rate (ρ) And The Logical Clock (*LocalTimer*)

Each node is driven by an independent, free-running local physical oscillator (i.e., the phase is not controlled in any way) and a logical-time clock (i.e., a counter), denoted *LocalTimer*, which locally keeps track of the passage of time and is driven by the local physical oscillator. An **oscillator tick**, also called a **clock tick**, is a discrete value and the basic unit of time in the network.

An ideal oscillator has zero drift rate with respect to real-time, perfectly marking the passage of time. Real oscillators are characterized by non-zero drift rates with respect to real-time. The oscillators of the nodes are assumed to have a known bounded drift rate, ρ, which is a small constant with respect to real-time, where ρ is a unitless non-negative real value and is expressed as $0 \leq \rho << 1$. The maximum drift of the fastest *LocalTimer* over a time interval of t is given by $(1+\rho)t$. The maximum drift of the slowest *LocalTimer* over a time interval of t is given by $(1/(1+\rho))t$. Therefore, the **maximum relative drift** of the fastest and slowest nodes with respect to each other over a time interval of t is given by the following equation.

$$\delta(t) = ((1+\rho) - 1/(1+\rho))t \quad (1)$$

2.2. Communication Delay (D), Network Imprecision (d), And γ

The communication latency between the nodes is expressed in terms of the minimum event-response delay, D, and network imprecision, d. These parameters have units of real time clock ticks and are described with the help of Figure 1. As depicted in this figure, a message

transmitted at real time t_0 is expected to arrive at all destination nodes, be processed, and subsequent messages are generated within the time interval of $[t_0+D, t_0+D+d]$. Communication between independently clocked nodes is inherently imprecise. The network imprecision, d, is the maximum time difference among all receivers of a message from a transmitting node with respect to real time. The imprecision is due to the drift of the oscillators with respect to real time, jitter, discretization error, temperature effects and differences in the lengths of the physical communication media. These two parameters are assumed to be bounded such that $D \geq 1$ and $d \geq 0$ and both have discrete values with units of real time clock tick. The communication latency, denoted γ, is expressed in terms of D and d, and is constrained by $\gamma = (D+d)$ and so has units of real time clock ticks.

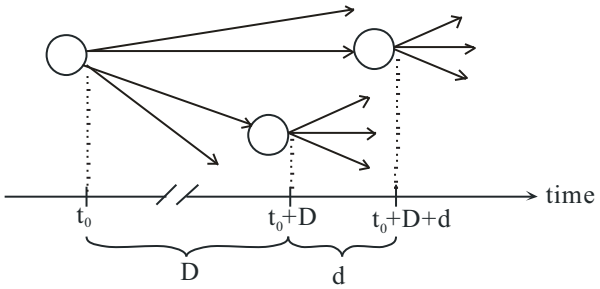

Figure 1. Event-response delay, D, and network imprecision, d.

2.3. Topology (*T*)

The general topology, T, considered is a strongly connected directed graph (digraph) consisting of K nodes, where each node is connected to the graph by at least one channel[1], there is a path[2] from any node to any other node, and the channels are either unidirectional or bidirectional. Furthermore, we assume there is no direct path from a node to itself, i.e., no self-loop, and there are no multiple channels directly connecting any two nodes in any one direction.

The number of strongly connected directed graphs for a given set of nodes have been studied by Liskovets [Lis 1970]. In this report, we use the terms network and graph interchangeably as well as the terms link, channel and edge. The following graph specific terms are used in the subsequent sections.

- Two nodes are said to be **adjacent** to each other or neighbors if they are connected to each other via a direct communication link.
- L, an integer value with units of links, denotes the largest **loop** in the graph, i.e., the maximum value of the longest path lengths[3] from a node back to itself visiting the nodes along the path only once (except for the first node which is also the last node).
- W, an integer value with units of links, signifies the **width** or diameter of the graph, i.e., the maximum value of the shortest path connecting any two nodes.

In general, for digraphs, L and W are at their maximum, i.e., $L = K$ and $W = K - 1$.

[1] A channel is an edge in the graph, a physical connection, between two nodes.
[2] A path is a logical connection consisting of one or more edges/links/channels.
[3] A path length is the number of edges/links/channels connecting any two nodes.

3. The Protocol

In this section we enumerate protocol assumptions, properties, parameters, and describe the protocol in pseudo-code. The general form of the distributed synchronization problem, *S*, is defined by the following septuple [Mal 2011].

$$S = (K, T, D, d, \rho, P, F)$$

In other words, the distributed synchronization problem is a function of the number of nodes (*K*), network topology (*T*), communication delay (*D*), communication imprecision (*d*), oscillator drift rate (ρ), synchronization period (*P*), and number of faults (*F*), respectively. The solution to this problem is a protocol with convergence and closure properties, at a minimum, as discussed subsequently in this section. However, in this protocol we do not deal with faults.

Each node is driven by an independent logical-time clock, i.e., *LocalTimer*. The clocks need to be periodically synchronized due to their inherent drift with respect to each other. In order to achieve synchronization, the nodes communicate by exchanging **Sync** messages. The periodic synchronization after achieving the initial synchrony is referred to as the **resynchronization process** whereby all nodes reengage in the synchronization process. A node is said to **time-out** when its *LocalTimer* reaches its maximum value. The resynchronization process begins when the first node (fastest node) times-out and transmits a *Sync* message and ends after the last node (slowest node) transmits a *Sync* message. For $\rho << 1$, the fastest node cannot time-out again before the slowest node transmits a *Sync* message [Mal 2011].

A node consists of a **synchronizer** and a set of **monitors**. A *Sync* message is transmitted either as a result of a resynchronization timeout, or when a node receives *Sync* message(s) indicative of other nodes engaging in the resynchronization process. The messages to be delivered to the destination nodes are deposited on communication channels.

The following definitions and terms are used in the description and operation of the protocol. All protocol parameters have discrete values with the time-based terms having units of real time clock ticks. The discretization is for practical purposes in implementing and model checking of the protocol. Although the network level measurements are real values, locally and at the node level, all parameters are discrete.

- The **resynchronization period**, denoted *P*, has units of real time clock ticks and is defined as the upper bound on the time interval between any two consecutive resets of the *LocalTimer* by a node.
- **Drift per *t***, denoted $\delta(t)$, has units of real time clock ticks and is defined as the maximum amount of drift between any two nodes for the duration of *t*, $\delta(t) \geq 0$. In particular:
  - Drift per *D*, denoted $\delta(D)$, for the duration of one *D*, $\delta(D) \geq 0$.
  - Drift per γ, denoted $\delta(\gamma)$, for the duration of one γ, $\delta(\gamma) \geq 0$.
  - Drift per *P*, denoted $\delta(P)$, for the duration of one period *P*, $\delta(P) \geq 0$.
- The **graph threshold**, T_S, is based on a specified graph topology and has units of real time clock ticks.

- The guaranteed precision or simply **precision** of the network, denoted π, $0 \leq \pi < P$, has units of real time clock ticks and is defined as the guaranteed achievable precision among all nodes.
- The **convergence time**, denoted C, has units of real time clock ticks and is defined as the bound on the maximum time it takes for the network to converge, i.e., to achieve synchrony.
- **Precision between *LocalTimers*** of any two adjacent nodes N_i and N_j at time t is denoted by $\Delta_{ij}(t)$ and has units of real time clock ticks.
- The **initial synchrony** is a state of the network and the earliest time when the precision among all nodes, upon convergence, is within π. The initial synchrony occurs at time C_{Init}.
- The **initial precision among *LocalTimers*** of all nodes at time t is by denoted $\Delta_{Init}(t)$, has units of real time clock ticks and is defined as a measure of the precision of the network after elapse time of C_{Init}.
- The **initial guaranteed precision** among *LocalTimers* of all nodes at time t is denoted by $\Delta_{InitGuaranteed}(t)$, has units of real time clock ticks and is a measure of the precision of the network after elapse time of C.

3.1. The Graph Threshold (T_S)

When a node receives a *Sync* message, except during a predefined window, it accepts the *Sync* message and undergoes the resynchronization process where it resets its *LocalTimer* and relays the *Sync* message to others. The predefined window where the node ignores all incoming *Sync* messages, referred to as **ignore window**, provides a means for the protocol to stop the vicious cycle of resynchronization processes triggered by the follow up *Sync* messages. The upper bound on the ignore window is referred to as the graph threshold, T_S, and is a function of a specified graph topology and satisfies the following equation.

$$T_S \geq (L+2)(\gamma + \delta(\gamma))$$

Defining T_S in terms of L requires knowledge of the topology of the given network. Thus, in order to generalize the expression for T_S, make it independent of the topology, and to help simplify the proof process, we express it in terms of its worst case value, $L = K$. However, for a specific application, optimizing T_S by expressing it in terms of L results in faster synchrony and better performance.

3.2. Sync Message And Its Validity

In order to achieve synchrony, the nodes communicate by exchanging *Sync* messages. Since only one message type is used for the operation of this protocol, a single bit suffices. When the system is in synchrony, the protocol overhead is at most one message per resynchronization period P. Assuming physical-layer error detections are dealt with separately, the reception of a *Sync* message is indicative of its validity in the value domain. The protocol performs as intended when the timing requirements of the messages from every node are satisfied. However, in the absence of faults, the reception of a *Sync* message is indicative of its validity in the value and time domains. A valid *Sync* message is discarded after it is relayed to the synchronizer and has been kept for one local clock tick.

3.3. The Monitor, The Synchronizer, And Protocol Functions

To assess the behavior of other nodes, a node employs as many monitors as the number of nodes it is connected to with one monitor for each source of incoming messages. A node neither uses nor monitors its own messages. A monitor keeps track of the activities of its corresponding source node. Specifically, a monitor reads, evaluates, validates, and stores the last valid message it receives from that node. Upon conveying the valid message to the local synchronizer, a monitor disposes of the valid message after it has been kept for one local clock tick.

The assessment results of the monitored nodes are utilized by the node in the synchronization process. The synchronizer describes the behavior of the node utilizing assessment results from its monitors.

The function *ValidateMessage()* used by the monitors determines whether a received *Sync* message is valid. We assume physical-layer error detections are dealt with separately. The function *ConsumeMessage()* used by the monitors invalidates the stored *Sync* message after it has been kept for one local clock tick. The function *ValidSync()* used by the synchronizer examines availability of valid *Sync* messages.

ValidateMessage():
if (incoming message = Sync) then
{Message is valid,
Store it.}

ConsumeMessage():
if (stored message timer > 1 tick) then
{Message is invalid,
Clear it.}

ValidSync():
if (number of stored messages > 0) then
{ return true,
else
return false.}

Figure 2. The protocol functions.

3.4. Protocol Assumptions

1. All nodes correctly execute the protocol.
2. All channels correctly transmit data from their sources to their destinations.
3. $K \geq 1$.
4. T = strongly connected digraph.
5. A message sent by a node will be received and processed by all other nodes within γ, where $\gamma = (D + d)$.
6. $0 \leq \rho << 1$.
7. Absence of faults in the links and nodes.
8. The initial values of the variables of a node are within their corresponding data-type range, although possibly with arbitrary values. (In an implementation, it is expected that some local mechanism exists to enforce type consistency for all variables.)

3.5. The Self-Stabilizing Distributed Clock Synchronization Problem

To simplify the presentation of this protocol, it is assumed that all time references are with respect to an initial real time t_0, where $t_0 = 0$ when the *protocol assumptions* are satisfied, and for all $t > t_0$ the system operates within the *protocol assumptions*.

We define the following symbols:

- C denotes a bound on the maximum convergence time,
- $\Delta_{Net}(t)$, for real time t, is the maximum difference of values of the *LocalTimers* of any two nodes (i.e., the relative clock skew) for $t \geq t_0$, and
- π, the synchronization precision, is the guaranteed upper bound on $\Delta_{Net}(t)$, for all $t \geq C$.

The maximum difference in the value of *LocalTimer* for all pairs of nodes at time t, $\Delta_{Net}(t)$, is determined by the following equation that accounts for the variations in the values of the *LocalTimer* across all nodes.

$$r = (W + 1)\gamma,$$
$$LocalTimer_{min}(x) = min\ (N_i.LocalTimer(x)), \text{ and}$$
$$LocalTimer_{max}(x) = max\ (N_i.LocalTimer(x)), \text{ for all } i,$$
$$\Delta_{Net}(t) = min\ ((LocalTimer_{max}(t) - LocalTimer_{min}(t)),$$
$$(LocalTimer_{max}(t - r) - LocalTimer_{min}(t - r))).$$

There exist C and π such that the following self-stabilization properties hold.

1. **Convergence:** $\Delta_{Net}(C) \leq \pi$, $0 \leq \pi < P$
2. **Closure:** For all $t \geq C$, $\Delta_{Net}(t) \leq \pi$
3. **Congruence:** For all nodes N_i, for all $t \geq C$, ($N_i.LocalTimer(t) = \gamma$ implies $\Delta_{Net}(t) \leq \pi$).

3.6. The Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs

The protocol is presented in Figure 3 and consists of a synchronizer and a set of monitors which execute once every local clock tick.

Synchronizer:	**Monitor:**
E1: if (*ValidSync()* and (*LocalTimer* < *D*)) *LocalTimer* := γ, E2: elseif ((*ValidSync()* and (*LocalTimer* ≥ T_S)) *LocalTimer* := γ, Transmit *Sync*, E3: elseif (*LocalTimer* ≥ *P*) // *time-out* *LocalTimer* := 0, Transmit *Sync*, E4: else *LocalTimer* := *LocalTimer* + 1.	**case (message from the corresponding node)** {*Sync*: *ValidateMessage()* *Other*: Do nothing. } // case *ConsumeMessage()*

Figure 3. The self-stabilizing clock synchronization protocol for arbitrary digraphs.

The following is a list of protocol parameters when all links are bidirectional.

$T_S \geq (L+2)(\gamma + \delta(\gamma))$
$P \geq 3T_S$, for $\rho = 0$
$P \geq 3T_S + 3\delta(T_S)$, for $L = K$ and $\rho > 0$
$P \geq \max((2K + 1)\gamma + (2K + 1)\delta(\gamma), 3T_S + 3\delta(T_S))$, for $L = f(T)$ and $\rho > 0$

The following is a list of protocol parameters for digraphs, i.e., when at least one link is unidirectional.

$T_S \geq (K+2)(\gamma + \delta(\gamma))$
$P \geq KT_S + K\delta(T_S)$

Regardless of the types of links in the network, the following is a list of protocol measures.

$C_{Init} = 2P + K(\gamma + \delta(\gamma))$
$\Delta_{Init}(C_{Init}) \leq (K - 1)(\gamma + \delta(\gamma))$
$C = C_{Init} + \lceil \Delta_{Init}(C_{Init}) / \gamma \rceil P$
$Wd \leq \Delta_{InitGuaranteed}(t) \leq W(\gamma + \delta(\gamma))$, for all $t \geq C$
$\pi = \Delta_{InitGuaranteed}(t) + \delta(P) \geq 0$, for all $t \geq C$, and $0 \leq \pi < P$

A trivial solution is when $P = 0$. Since $P > T_S$ and the *LocalTimer* is reset after reaching P (worst-case wraparound), a trivial solution is not possible.

Appendix B provides an example to give the reader a quick review and help in understanding of the behavior of the protocol.

4. Verification Model

There are two general formal methods approaches for the verification of the correctness of a protocol; **theorem proving** and **model checking**. Verification via theorem proving requires a deductive proof of the protocol. Verification via model checking is based on specific scenarios and generally limited to a subset of the problem space. A deductive proof of the protocol will be the subject of a subsequent report. In the mean time, we chose the model checking approach for its ease, feasibility, and quick examination of a subset of the problem space while attempting a more comprehensive proof via theorem proving. In this section, we present the details of the model checking efforts by describing models of the system components, their data structures, and the modeling simplification and abstractions techniques employed in the mechanical verification of the protocol.

A matter of concern in model checking is the ease of encoding the algorithm and assumed environment in the language of the model checker. In model checking, the state explosion, i.e., the time and space required to run the model checker, grows rapidly and eventually becomes infeasible as the size and complexity of the model grows. Therefore, abstraction must be employed with respect to the size of the model and real-time delays.

The algorithm described in this report is fairly subtle and must cope with many kinds of timing behaviors. Model checking has been used to explore and verify distributed algorithms but faces certain difficulties [Ste 2004] [Lön 1997] [Mal 2008]. One of the foremost challenges is a realistic representation of time as a continuous variable. Timed automata provide a suitable formalism of this kind and are mechanized in model checkers such as Kronos and UPPAAL [Ste 2004]. Model checking for timed automata is computationally complex. As the network size and complexity increase, the resulting state explosion renders the model computationally infeasible.

As we elaborated earlier in this report, although the network level measurements are real values, locally and at the node level, all parameters are discrete. Since continuous time model is impracticable, we looked for an abstraction employing discrete time. Also, although we cannot yet prove the soundness of this abstraction, our decision to use a discrete model for time was critical to our ability to undertake this verification effort. A basic model of various elements of the protocol is presented Figure 4 where the concurrent operations are separated by '||'.

The Symbolic Model Verifier (SMV) was used in modeling of this protocol on a PC with 4GB of memory running Linux [SMV]. SMV allows the designers to formally verify temporal logic properties of finite state systems. Developers use SMV to verify the design for all possible input sequences, instead of a chosen selection of sequences as in simulation. SMV's language description and modeling capability provide relatively easy translation from the pseudo-code. SMV also provides the desired capability to introduce randomness into the initial values of the variables.

SMV syntax consists of a hierarchy of modules. Modules can be instantiated many times, where each instantiation creates a copy of the local variables. The parameters to a module are passed

by reference. SMV semantics is synchronous composition, where all assignments are executed in parallel and synchronously. Thus, a single step of the resulting model corresponds to a step in each of the components. SMV is also a parallel assignment language with guarded assignments. Guards are evaluated sequentially. The first one that is true determines the resulting value and if none of the guards are true, result is numeric value 1.

```
Global Constants K, D, d, P, T_S, γ, maxMessageTimer : integer
MessageType                                          : {NONE , Sync}

MonitorType (InputMessage, ValidatedMessage)
{
    Input     InputMessage        : MessageType
    Output    ValidatedMessage    : Boolean

    if (InputMessage = Sync)
        ValidatedMessage := True
    else
        ValidatedMessage := False
}

SynchronizerType (ValidatedMessages, TransmitMessage)
{
    Input     ValidatedMessages   : array of Boolean [NumInputs]
    Output    TransmitMessage     : MessageType
    Local     LocalTimer          : integer, range = 0 .. P

    Function  ValidSync() := OR (ValidatedMessages[j]), j = 1 .. NumInputs

    TransmitMessage := NONE

    if (ValidSync() and (LocalTimer < D))
        LocalTimer := γ
    elseif (ValidSync() and (LocalTimer ≥ T_S))
        LocalTimer := γ
        TransmitMessage := Sync
    elseif (LocalTimer ≥ P))
        LocalTimer := 0
        TransmitMessage := Sync
    else
        LocalTimer := LocalTimer + 1
}
```

Figure 4.a. The basic model.

```
NodeType (NumInputs, InputMessages, NumOutputs, OutputMessages)
{
    Input       InputMessages          : array of MessageType [NumInputs]
    Output      OutputMessages  : array of MessageType [NumOutputs]
    Local       Monitors               : array of MonitorType [NumInputs]
                Synchronizer           : SynchronizerType
                TransmitMessage: MessageType
                ValidatedMessages      : array of Boolean [NumInputs]

    Synchronizer(ValidatedMessages, TransmitMessage)

    OutputMessages[1] := TransmitMessage ||
    OutputMessages[2] := TransmitMessage ||
     ..          ..              ..                ||
    OutputMessages[h] := TransmitMessage, h = 1 .. NumOutputs

    Monitors[1](InputMessages[1], ValidatedMessages[1]) ||
    Monitors[2](InputMessages[2], ValidatedMessages[2]) ||
     ..          ..              ..                ..                ||
    Monitors[j](InputMessages[j], ValidatedMessages[j]), j = 1 .. NumInputs
}

Network ::
{
    Local       Nodes                  : array of NodeType [K]

    Loop forever ()
        Nodes[1] || Nodes[2] || .. || Nodes[K]
}
```

Notation: Concurrent operations are separated by '||'.

Figure 4.b. The basic model.

4.1. Modeling Communication Channels

An explicit model of the communication channel requires a separate entity (SMV module) with its own local memory, at a minimum, to store and forward a message. This approach would readily exhaust the available 4GB memory even for small values of K and render the model checking effort ineffective. To reduce state space, channels are implicitly modeled and the outgoing message is kept within the transmitting node long enough for the receiving nodes to sample it.

4.2. Modeling Monitors

A monitor keeps track of activities of its corresponding source node and manages message validity. Recall that we assume physical-layer error detections are dealt with separately and so, receiving a *Sync* message is indicative of its validity in the value and time domains. In other words, we analyze the system at the point where the valid messages arrive at the *Synchronizer* of the node. Since we assume no faulty nodes are present, an explicit model of the monitors is not necessary. Instead, and to reduce the state space, monitors are implicitly modeled at the receiving nodes.

4.3. Modeling Nodes

The synchronizer describes the collective behavior of the node utilizing assessment results from its monitors. The local measures within each node are used to keep track of timing of the self-stabilization events. Although the protocol parameters are defined with respect to real time, ultimately, in implementations they have to be translated into discrete values. Discretization of the protocol parameters is performed using the ceiling operation. In this protocol, all local variables and watchdog timers are discretized and represented by integer values. These local variables are, therefore, measured with respect to the local clock.

A parameterized node, *NodeType*, is introduced that executes the protocol and consists of local variables. The *NodeType*'s data structure consists of *Monitors*, *Synchronizer*, and *MessageOut*. The *Synchronizer* in turn consists of *LocalTimer* which represents the duration of time since the node has gone through the resynchronization process. The *MessageOut* element represents the out going message of the node. The range of values that these elements can hold is enumerated as follows.

$$LocalTimer = \{0 \,..\, P\}$$
$$MessageOut = \{NONE,\ Sync\}$$

In the basic models of Figure 4 system parameters, e.g., K, P, and T_S are defined as global constants. However, in the SMV implementation some of these parameters are passed on to the node as input parameters. In particular, the parameters T_S and P are customized for each node and are passed on to the node as input parameters (Section 4.6). Also, the SMV implementation of the *NodeType* has an additional input parameter, *NodeId*, that is not a protocol requirement but is used in the model checking process for node-specific operations, e.g., to specify a node with the highest drift rate, i.e., the fastest/slowest node.

The set of unidirectional inputs/outputs links, $InputMessages_j$/$OutputMessages_h$, specify the input/output links and sources/destination of the messages, respectively. Together, they define the network topology.

Because of the message validity assumptions and implicit model of the monitors, the related protocol functions are implemented at the *NodeType*. These functions examine the number of available messages at the transmitting node utilizing implicit model of the communication channels. The function *ValidSync()* is an *or* operation over the set of input messages.

$$ValidSync() = OR\ (Node_j.MessageOut),\ i \neq j$$

4.4. Modeling Communication Delays

Since we have assumed absence of malicious faulty nodes, the nodes react to each other's messages within γ and the minimum event-response delay, D, and the network imprecision, d, do not play distinctive roles in the synchronization process. In other words, the effects of D and d in the synchronization process are incorporated in γ. This assertion is not true in the presence of malicious faulty nodes. These parameters, however, directly contribute to the guaranteed precision of the network.

An explicit model of D and d requires more memory to store and delay a message both in the node and the communication channel modules. These explicit models would exponentially increase state space. Recall that all system parameters are discretized to local ticks. Therefore, an increase of one local tick in the communication delays directly increases the value of all other timing parameters. As a result, this approach would readily exhaust the available 4GB memory even for small values of K and render the model checking effort ineffective. To further minimize state space, D and d are chosen to be at their minimum values of 1 and 0 clock ticks, respectively. As a result, γ is at its minimum value of 1 clock tick. This simplification, consequently, implies that the local oscillators of the nodes are in phase with each other but it does not imply that the nodes are synchronized with each other.

4.5. Modeling Clocks and Timers

Each node has a logical clock, *LocalTimer*, that locally keeps track of time. This logical clock is used in measuring the self-stabilization precision, π, across the nodes from an external view of the system. A single clock per node suffices to advance a nodes's *LocalTimer*. Since $\gamma = 1$ clock tick, a single clock suffices to advance all *LocalTimers*. To further minimize the state space, all timers, *LocalTimers* and *GlobalClock* (Section 4.7), are incremented once per model checker cycle. The SMV cycle, therefore, binds the whole system together, providing a means for advancing the *GlobalClock* and the *LocalTimer* at the nodes and providing an external view of the system at any time. Although the use of SMV cycle, along with $\gamma = 1$ clock tick, does not imply synchrony at the nodes, it does imply that the nodes are in phase with each other at the local oscillator level. However, due to the inherent non-deterministic execution of a model in the model checker, the order of execution of the nodes is not predetermined. Since there is no control over the order of transmission of messages and the start of execution of the nodes at each model checker cycle, the nodes potentially broadcast and receive messages out of order of issuance.

4.6. Modeling Drift

In a realizable distributed system the clocks drift with respect to real time and each other. As a result, any viable solution has to account for the clock drift rate, ρ. An explicit model of ρ would require dealing with real values. Since all parameters are locally represented with integer values, we opt to stretch the time line by an equivalent factor in order to deal with integer values instead. Dealing either with real values or their equivalent integer values for ρ increases the state space drastically. As a result, this approach would readily exhaust the available 4GB memory even for small values of K and render the model checking effort ineffective.

To reduce state space, we have introduced a modeling technique to model ρ implicitly. Hereafter, we refer to it as **implicit drift model**. We explain this modeling technique with respect to P. In the absence of drift, all nodes have the same synchronization period, P. In the presence of drift, the effective synchronization period of a node is a function of the drift rate associated with the node (its local oscillator). The relative drift of any two nodes is also a function of their associated drift rates with at least two nodes in the system having the maximum relative drift of $\delta(P)$ during the synchronization period. In *implicit drift model* approach, instead of explicitly specifying the drift rate for a node's local oscillator and determining the node's drift on a clock tick base, we determine the node's effective period based on the drift rate and pass the effective period to the node. Since the synchronization period is passed down to a node as an input parameter, each node will have its unique synchronization period with the incorporated amount of drift. In this approach the effective synchronization period is directly applied to the nodes with at least one node being the slowest and another the fastest in the system with their maximum relative drift being $\delta(P)$. One advantage of this modeling technique is that it drastically reduces state space. Another advantage is that when a node's behavior is not influenced by the behavior of other nodes for duration of time, the model checking time can advance to the end of that time interval[4]. Thus, the *implicit drift model* substantially improves the model checking performance.

In order to expedite the self-stabilization process, in general, and in order to minimize the state space for model checking purposes, in particular, the convergence time has to be minimized. The convergence time is a function of P, therefore, P has to be as small as possible. Since P is a function of T_S ($T_S < P$), T_S too has to be optimized. When state space exceeds our available resources, we optimize P by expressing T_S in terms of the largest loop, L, for the network (instead of its default maximum value) that is being model checked.

We apply the *implicit drift model* approach to all parameters that are based on time including γ, T_S, and P. The amount of drift applied to a particular parameter is linearly proportional to its value. Since typically $\rho << 1$ and γ is very small, the effect of ρ during γ is negligible, i.e., $\delta(\gamma) = 0$. Also, since all parameters are locally defined as integers, we set T_S and P to large enough values, beyond their minimum values, to guarantee proportional presence of the effect of drift in T_S and P in the nodes.

[4] The concept of advancing time has been used in hardware description language (e.g., VHDL and Verilog) simulation tools for decades.

In the advent of a more capable model checker and availability of more memory to handle explicit models of D, d and the communication channels, the *implicit drift model* is still a viable technique.

As mentioned earlier, the use of SMV cycle, along with $\gamma = 1$ clock tick, imply that the nodes are in phase with each other at the local oscillator level. However, applying the *implicit drift model* implies that the nodes are out of phase with each other at the *LocalTimer* level. Once again, due to the inherent non-deterministic execution of a model in the model checker, the order of execution of the nodes is not predetermined, there is no control over the order of transmission of messages and the start of execution of the nodes at each model checker cycle, thus, the nodes potentially broadcast and receive messages out of order of issuance. As a result, we believe our modeling techniques and abstractions properly capture the intended properties of a realizable system.

4.7. Modeling Network

Model checking is conducted on a given network consisting of a set of nodes that are instances of the *NodeType* and are interconnected to reflect a desired topology. A single step of the resulting model corresponds to a step in each of the components. A global clock, *GlobalClock*, is introduced to measure passage of time from the beginning of the operation and with respect to the real time and from the perspective of an external observer. The *GlobalClock* is used to measure the convergence time, C, and is incremented once per model checker cycle.

The synchronization properties are examined at the network level and provide an external view of the system. The properties examined to verify the claims of the protocol are described in section 5.

5. Propositions

Computational tree logic (CTL), a temporal logic, is used to express properties of a system in this context. CTL uses atomic propositions as its building blocks to make statements about the states of a system. CTL then combines these propositions into formulas using logical and temporal operators with quantification over runs. In CTL formulas are composed of **path quantifiers**, *E* and *A*, and **temporal operators**, *X*, *F*, *G*, and *U* [Cla 1981].

Symbol	Meaning
E	there exists an execution
A	for all executions
X	next
F	finally (eventually)
G	globally

In this section the claims of convergence, closure, and congruence properties as well as the claims of maximum convergence time and determinism of the protocol are examined. Although in the description of the protocol convergence and closure properties are stated separately, they are examined via one CTL proposition. This proposition also expresses the claims of determinism and linear convergence. Validation of this general CTL proposition requires examination of a number of underlying propositions. In particular, since $\Delta_{LocalTimer}(t)$ is defined in terms of the *LocalTimer* of the nodes, examination of the properties that described proper behavior of the *LocalTimer* take precedence. In this section, the general propositions that verify the convergence, closure, and congruence properties of the protocol as well as the claims of maximum convergence time and determinism are examined.

The variable *ElapsedTime* is used in these properties and is defined here.

ElapsedTime = (GlobalClock >= ConvergenceTime) ;

The *GlobalClock* is a measure of elapsed time from the beginning of the operation and with respect to the real time, i.e., external view. The *ElapsedTime* is indicative of the *GlobalClock* reaching its target maximum value of *ConvergenceTime*.

Proposition *Liveness*: This property addresses the *Liveness* property of the system by examining whether or not time advances and the amount of time elapsed, *ElapsedTime*, has advanced beyond the predicted convergence time, *ConvergenceTime*.

AF (ElapsedTime)

Proposition *ConvergenceAndClosure*: This proposition encompasses the criteria for the convergence and the closure properties as well as the claims of maximum convergence time and determinism. This proposition specifies whether or not the system will converge to the predicted precision after the elapse of convergence time, *ElapsedTime*, and whether or not it will remain within that precision thereafter. This and subsequent properties are expected to hold.

$$
\begin{array}{ll}
AF\ (ElapsedTime) \wedge & \text{-- Determinism Property} \\
AG\ (ElapsedTime \rightarrow AllWithinPrecision) \wedge & \text{-- Convergence Property} \\
AG\ ((ElapsedTime \wedge AllWithinPrecision) \rightarrow & \\
\quad AX\ (ElapsedTime \wedge AllWithinPrecision)) & \text{-- Closure Property}
\end{array}
$$

The proper value of the *AllWithinPrecision* is determined by measuring the difference of maximum and minimum values of the *LocalTimers* of all nodes for the current tick and in conjunction with the result from the previous $(W+1)\gamma$ ticks. The expected difference of *LocalTimers* is the predicted precision bound.

To eliminate trivial results and false positives, the following proposition is examined and the expected result is a false value. This property specifies that after the elapse of convergence time, *ElapsedTime*, whether or not the system will not converge or if it converges, whether or not it drifts apart beyond the expected precision bound.

$$
\begin{array}{l}
AF\ (ElapsedTime) \wedge \\
AG\ (ElapsedTime \rightarrow AllWithinPrecision) \wedge \\
AG\ ((ElapsedTime \wedge AllWithinPrecision) \rightarrow EX\ (\neg AllWithinPrecision))
\end{array}
$$

Proposition *Congruence*: This property specifies the criteria for the congruence property of the protocol. This property is described with respect to only one node, namely *Node_1*. Since all nodes are identical, due to symmetry, the result of the proposition equally applies to other nodes.

$$
\begin{array}{ll}
AF\ (ElapsedTime) \wedge & \\
AG\ ((ElapsedTime \wedge (Node\_1.LocalTimer = \gamma)) \rightarrow & \\
\quad AX\ (ElapsedTime \wedge AllWithinPrecision)) & \text{-- Congruence Property}
\end{array}
$$

6. Results And Conclusion

Since in the protocol we do not limit the size of the network, K, model checking of all possible digraphs for all K, even for idealized scenarios ($d = 0$, $\rho = 0$), is simply impossible. Model checking of all possible topologies for a given K is also a daunting task. Given the limited resources available and to circumvent state space explosion, we had to limit the network size. Nevertheless, to verify our claims of the correctness of the protocol, we have model checked all possible digraphs for smaller K. Additionally, we were able to model check some topologies for larger K. Table 1 is a list of the model checked networks with their sizes and corresponding number of topologies while bounding the drift to $0 \leq \rho \leq 0.2$. Each row of the table corresponds to a given K and two types of topologies considered with the number of model checked graphs of the possible total combinations for the corresponding topology type in its column.

Table 1. Model checked networks.

K	**Topology** (all links bidirectional)	**Topology** (digraphs)
2	1 of 1	1 of 1
3	2 of 2	5 of 5
4	6 of 6	83 of 83
5	21 of 21	Single Directed Ring 2 Variations of Doubly Connected Directed Ring
6	112 of 112	-
7	Linear*	Linear*
7	Star*	Star*
7	Fully Connected*	Fully Connected*
7 (3×4)	Fully Connected Bipartite*	Fully Connected Bipartite*
7	Combo	4 of 4
7	Grid	-
7	Full Grid	-
9 (3×3)	Grid	-
15	Star*	Star*
20	Star*	Star*

* For *Linear* and *Star* topologies and for the network to be strongly connected (to be precise,1-connected), the links are by necessity bidirectional. For *Fully Connected (Complete)* and *Fully Connected Bipartite* topologies the links are by definition bidirectional.

One example of a random graph is depicted in Figure 5. The *Combo* topology is a 7-node graph consisting of a *Linear* topology of two nodes (1 and 2), a *Ring* topology of three nodes (2, 3, and 4), and a *Star* topology of four nodes (4, 5, 6, and 7). Note that there is only one possible digraph for the *Linear* and *Star* topologies. Also, for three nodes, there are five digraphs.

However, for a *Ring* of three nodes, there are four variations. Therefore, after omitting symmetry, there are four digraphs for the *Combo* topology to be examined. Sample SMV codes will be made available at (http: //shemesh.larc.nasa.gov/people/mrm/publicantion.htm).

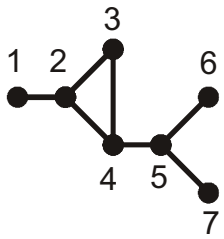

Figure 5. Combo topology.

A bounded model of *A Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs* is model checked using SMV where, for a set of digraphs, the entire state space is examined and verified to self-stabilize from an arbitrary state. This SMV model checking effort was performed on a PC with 4GB of memory running Linux. We described modeling concepts by abstracting the problem to discrete time and for realizable systems. The model checking results have confirmed the correctness of the protocol as they apply to the networks with unidirectional and bidirectional links as described earlier (Section 2.3). Also, the results indicate that the protocol is applicable to realizable systems and practical applications. In addition, the results confirmed the claims of determinism and linear convergence with respect to the synchronization period. Because of the model checking results, we conjecture that the protocol solves the general case of this problem for all $K \geq 1$ and is applicable to realizable systems and practical applications. Furthermore, this model checking effort has shown that, at a minimum, a deterministic solution for this problem exists.

References:

[But 2008] Butler, R.: "A primer on architectural level fault tolerance," NASA/TM-2008-215108, February 2008.

[Cla 1981] Clarke, E.M.; Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In Logic of Programs: Workshop, Yorktown Heights, NY, May 1981, LNCS 131. Springer, 1981.

[Gir 2005] Girault, A.; Rutten, E.: "Modeling Fault-tolerant Distributed Systems for Discrete Controller Synthesis," Electronic Notes in Theoretical Computer Science, vol. 133, pp. 81-100, 2005.

[Kop 1997] Kopetz, H: "Real-Time Systems, Design Principles for Distributed Embedded Applications," Kluwar Academic Publishers, ISBN 0-7923-9894-7, 1997.

[Lis 1970] Liskovets, V.A.: "number of strongly connected directed graphs," Matmaticheskie Zameki, Vol. 8, No. 6, pp. 721-732, December 1970.

[Lön 1997] Lönn, H.; and Pettersson, P.: "Formal verification of a TDMA protocol start-up mechanism" In Pacific Rim International Symposium on Fault-Tolerant Systems, pages 235–242, Taipei, Taiwan, Dec. 1997. IEEE Computer Society.

[Mal 2006] Malekpour, M.R.: A Byzantine-Fault Tolerant Self-Stabilizing Protocol for Distributed Clock Synchronization Systems. Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS06), November 2006.

[Mal 2008] Malekpour, M.R.: "Verification of a Byzantine-Fault-Tolerant Self-Stabilizing Protocol for Clock Synchronization." IEEE Aerospace Conference, March 2008.

[Mal 2011] Malekpour, M.R.: "A Self-Stabilizing Distributed Clock Synchronization Protocol For Arbitrary Digraphs". NASA/TM-2011-217054, pp. 42, February 2011.

[SMV] http://www-2.cs.cmu.edu/~modelcheck/smv.html

[Ste 2004] Steiner, W.; Rushby, J.; Sorea, M.; Pfeifer, H.: "Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration To Exhaustive Fault Simulation," The International Conference on Dependable Systems and Networks (DSN'04), 2004.

[Tor 2005] Torres-Pomales, W; Malekpour, M.R.; Miner, P.S.: ROBUS-2: A fault-tolerant broadcast communication system. NASA/TM-2005-213540, pp. 201, March 2005.

Appendix A. Symbols

The symbols used in the protocol are described in detail in [Malekpour 2010] and are listed here for reference.

Symbols	Descriptions
K	sum of all nodes
T	network topology
D	event-response delay
d	network imprecision
ρ	bounded drift rate with respect to real time
P	self-stabilization/synchronization period
F	sum of all faulty nodes
N_i	the i^{th} node
M_i	the i^{th} monitor of a node
γ	communication latency
L	the largest loop in the graph
W	the width or diameter of the graph
T_S	graph threshold
π	the guaranteed self-stabilization/synchronization precision
C	convergence time
C_{Init}	time of initial synchrony
LocalTimer	node's local logical clock
$\Delta_{ij}(t)$	precision between *LocalTimers* of any two adjacent nodes N_i and N_j at time t
$\Delta_{Init}(t)$	initial precision among *LocalTimers* of all nodes at time t
$\Delta_{InitGuaranteed}(t)$	initial guaranteed precision among *LocalTimers* of all nodes at time t
$\delta(t)$	drift per t
Sync	self-stabilization/synchronization message
$\Delta_{Net}(t)$	precision among *LocalTimers* of all nodes at time t

Appendix B. Example

The purpose of this example is to give the reader a quick review and help in understanding of the behavior of the protocol. The following is an example of a ring topology consisting of 5 nodes, interconnected with bidirectional links, operating under the ideal conditions. In the absence of clock drift (i.e., ideal condition), $\rho = 0$, we abstracted the communication delay to 1 clock tick. Table B.1 is an execution trace of the system and has seven columns; one for time reference, one for each node, and the last column for network precision, π. Each depicts activities of all nodes at the corresponding time. Cell contents for the node columns consist of a number corresponding to the value of the *LocalTimer* of the node in conjunction with a *Sync* message if the node transmits the message.

System parameters:

$K = 5$ nodes ➔ $W = \lceil K / 2 \rceil = 3$ nodes

$\rho = 0$ ➔ $\delta(\gamma), \delta(P), \delta(*) = 0$

$D = 1$ clock tick, $d = 0$ clock tick ➔ $\gamma = 1$ clock tick

$T_S \geq (K+2)(\gamma + \delta(\gamma)) = (5 + 2)(1 + 0) = 7$clock ticks

$P \geq KT_S + K\delta(T_S) = 5 * 7 + 0 = 35$ clock ticks

$C_{Init} = 2P + K(\gamma + \delta(\gamma)) = 2 * 35 + 5(1 + 0) = 80$ clock ticks

$\Delta_{Init}(C_{Init}) \leq (K - 1)(\gamma + \delta(\gamma)) = (5 - 1)(1 + 0) = 4$ clock ticks

$C = C_{Init} + \lceil \Delta_{Init}(C_{Init}) / \gamma \rceil P = 80 + 4/1 = 84$ clock ticks

$Wd \leq \Delta_{InitGuaranteed}(t) \leq W(\gamma + \delta(\gamma))$, for all $t \geq C$ ➔ $\Delta_{InitGuaranteed}(t) = 0$ clock tick

$\pi = \Delta_{InitGuaranteed}(t) + \delta(P) \geq 0$, for all $t \geq C$, and $0 \leq \pi < P$ ➔ $\pi = 0$ clock tick

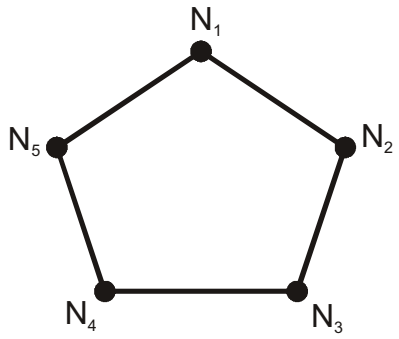

Figure B.1. A ring topology.

Table B.1. An execution trace of a ring of 5 nodes.

Time	N_1	N_2	N_3	N_4	N_5	π
1	22	4	33	25	2	31
2	23	5	34	26	3	31
3	24	6	0, *Sync*	27	4	24
4	25	7	1	1, *Sync*	5	24
5	26	8	2	2	6	24
…	…	…	…	…	…	…
13	34	16	10	10	14	24
14	0, *Sync*	17	11	11	15	17
15	1	1, *Sync*	12	12	1, *Sync*	11
16	2	2	1, *Sync*	1, *Sync*	2	1
17	3	3	2	2	3	1
18	4	4	3	3	4	1
19	5	5	4	4	5	1
20	6	6	5	5	6	1
…	…	…	…	…	…	…
48	34	34	33	33	34	1
49	0, *Sync*	0, *Sync*	34	34	0, *Sync*	1
50	1	1	1, *Sync*	1, *Sync*	1	0
51	2	2	2	2	2	0
…	…	…	…	…	…	…

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 05 - 2011	Technical Memorandum	

4. TITLE AND SUBTITLE

Model Checking a Self-Stabilizing Distributed Clock Synchronization Protocol for Arbitrary Digraphs

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Malekpour, Mahyar R.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

534723.02.02.07.30

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

L-20029

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/TM-2011-217152

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 62
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report presents the mechanical verification of a self-stabilizing distributed clock synchronization protocol for arbitrary digraphs in the absence of faults. This protocol does not rely on assumptions about the initial state of the system, other than the presence of at least one node, and no central clock or a centrally generated signal, pulse, or message is used. The system under study is an arbitrary, non-partitioned digraph ranging from fully connected to 1-connected networks of nodes while allowing for differences in the network elements. Nodes are anonymous, i.e., they do not have unique identities. There is no theoretical limit on the maximum number of participating nodes. The only constraint on the behavior of the node is that the interactions with other nodes are restricted to defined links and interfaces. This protocol deterministically converges within a time bound that is a linear function of the self-stabilization period.

15. SUBJECT TERMS

Algorithm, Clock Synchronization, Communication Network, Digraphs, Distributed Systems, Protocol, Self-Stabilizing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	31	(443) 757-5802